实验 “感知机”的Python程序实现

**[实验目的]：**掌握在jupyter环境下的编程方法；掌握python类的定义；掌握逻辑门的程序实现方法；理解感知机的工作原理。

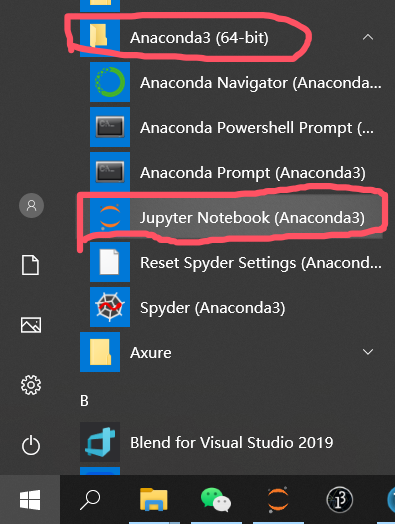
**[实验软件]** anaconda、 jupyter

**[实验内容]**

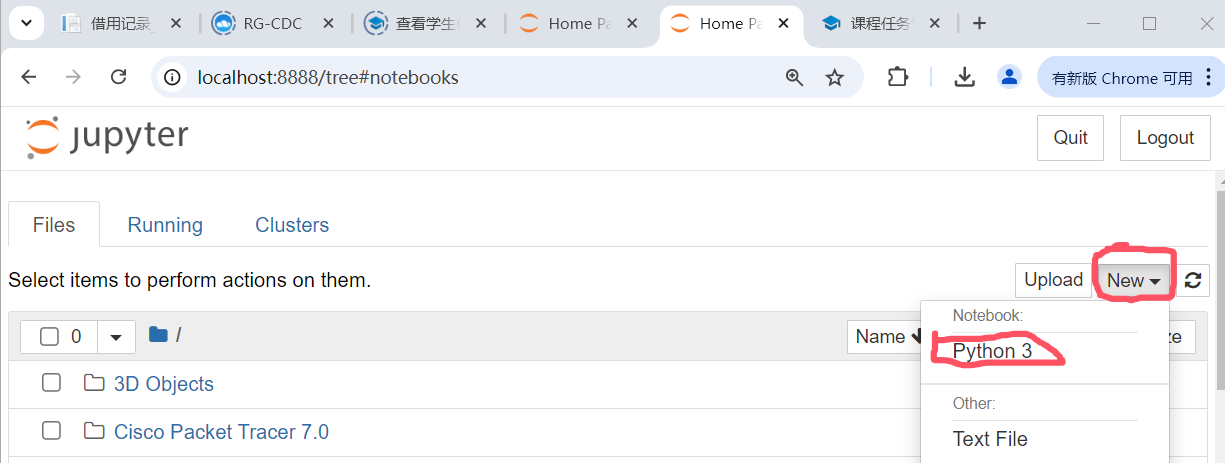
在jupyter环境下根据“第2章 感知机.pdf”内容，定义perceptron类（class），类中包含如下函数定义（def）：与门（AND）、或门（OR）、与非门（NAND）和异或门（XOR），并在后续的cell中实现调用并验证结果。

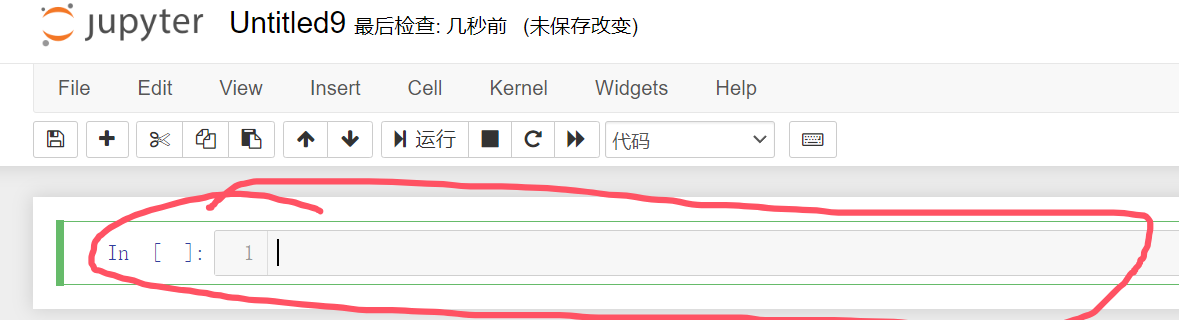
**[实验指导]**

1. 打开Jupyter ，会出现一个黑框，黑框不能关闭。

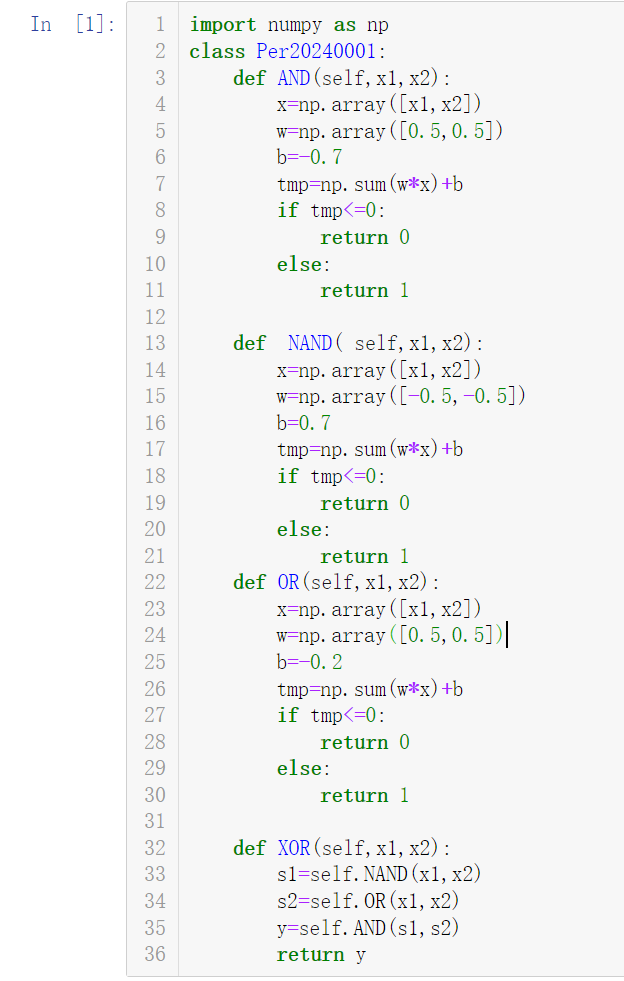


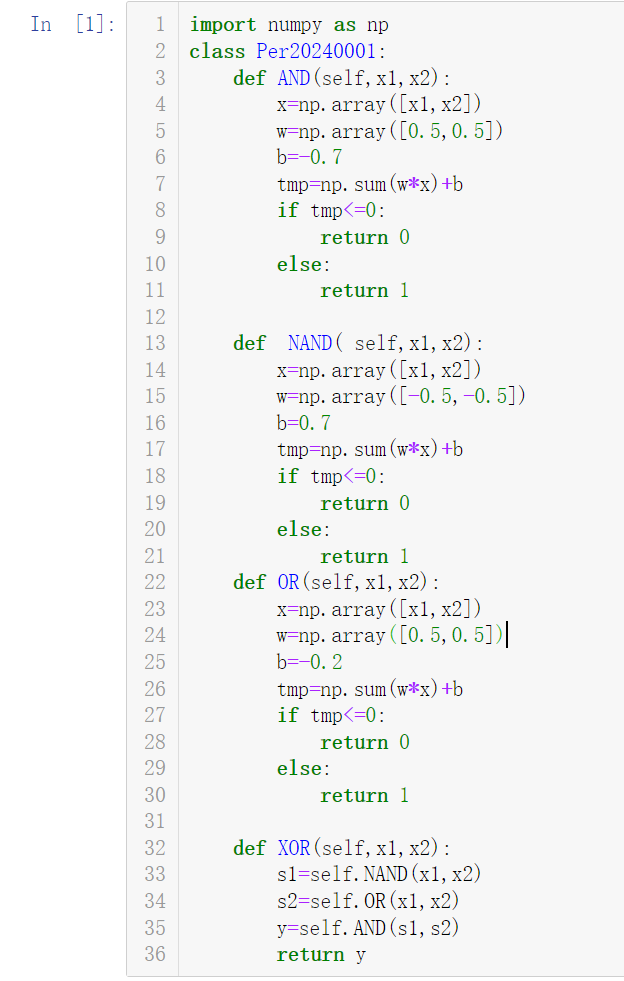
1. 新建程序文件





1. 定义感知机类，相关示例代码如下。

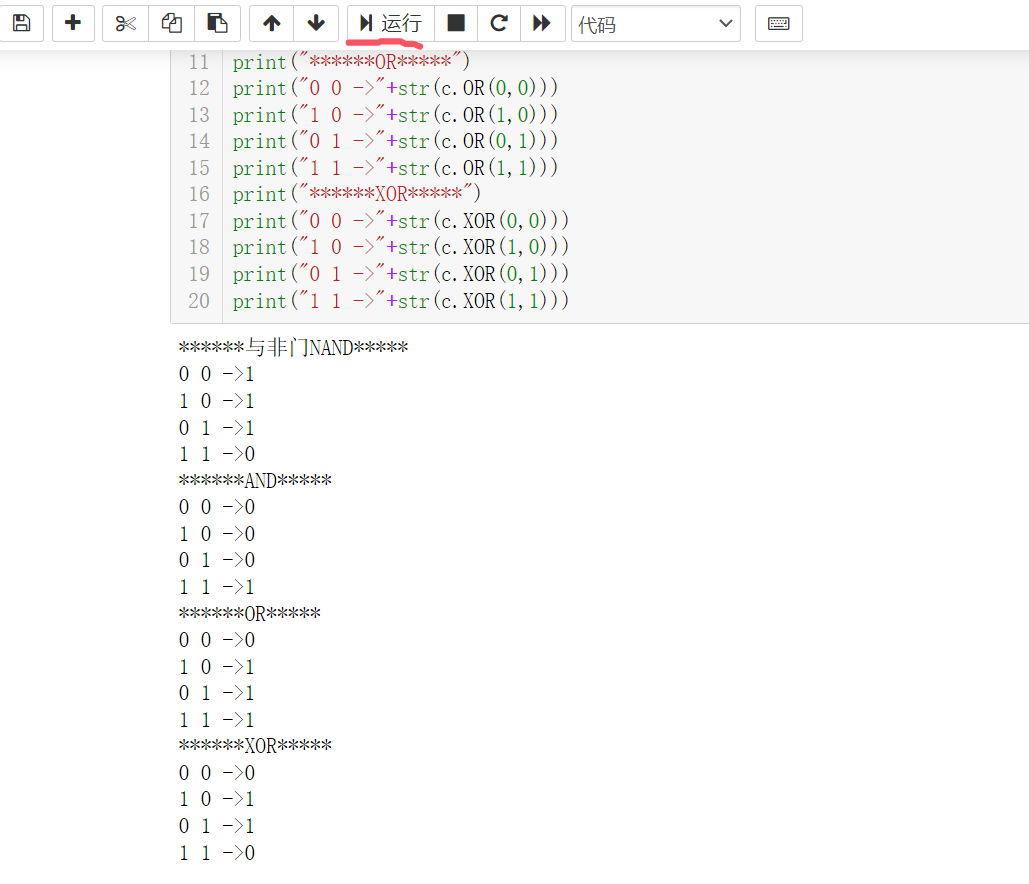




1. 用感知机类定义对象、测试验证与门（AND）、或门（OR）、与非门（NAND）和异或门（XOR），相关代码如下。



1. 运行程序查看结果。



【实验报告】根据上面的实验指导，结合教材为实验指导中的代码添加注释，类名Per20240001需要改为自己的学号，将加了注释的代码和运行结果放入下面。

添加注释后的代码:

import numpy as np

class Per202306630:

    """

    一个简单的感知器类，用于实现基本的逻辑运算（AND, NAND, OR, XOR）。

    """

    def AND(self, x1, x2):

        """

        实现逻辑 AND 运算。

        参数：

            x1 (int): 第一个输入值 (0 或 1)。

            x2 (int): 第二个输入值 (0 或 1)。

        返回：

            int: AND 运算的结果 (0 或 1)。

        """

        x = np.array([x1, x2])  *# 将输入转换为 NumPy 数组*

        w = np.array([0.5, 0.5])  *# 定义权重数组*

        b = -0.7  *# 定义偏置*

        tmp = np.sum(w \* x) + b  *# 计算加权和加上偏置*

        if tmp <= 0:

            return 0  *# 如果加权和小于等于 0，返回 0*

        else:

            return 1  *# 否则，返回 1*

    def NAND(self, x1, x2):

        """

        实现逻辑 NAND (与非) 运算。

        参数：

            x1 (int): 第一个输入值 (0 或 1)。

            x2 (int): 第二个输入值 (0 或 1)。

        返回：

            int: NAND 运算的结果 (0 或 1)。

        """

        x = np.array([x1, x2])  *# 将输入转换为 NumPy 数组*

        w = np.array([-0.5, -0.5])  *# 定义权重数组*

        b = 0.7  *# 定义偏置*

        tmp = np.sum(w \* x) + b  *# 计算加权和加上偏置*

        if tmp <= 0:

            return 0  *# 如果加权和小于等于 0，返回 0*

        else:

            return 1  *# 否则，返回 1*

    def OR(self, x1, x2):

        """

        实现逻辑 OR 运算。

        参数：

            x1 (int): 第一个输入值 (0 或 1)。

            x2 (int): 第二个输入值 (0 或 1)。

        返回：

            int: OR 运算的结果 (0 或 1)。

        """

        x = np.array([x1, x2])  *# 将输入转换为 NumPy 数组*

        w = np.array([0.5, 0.5])  *# 定义权重数组*

        b = -0.2  *# 定义偏置*

        tmp = np.sum(w \* x) + b  *# 计算加权和加上偏置*

        if tmp <= 0:

            return 0  *# 如果加权和小于等于 0，返回 0*

        else:

            return 1  *# 否则，返回 1*

    def XOR(self, x1, x2):

        """

        实现逻辑 XOR (异或) 运算，通过组合 NAND, OR 和 AND 实现。

        参数：

            x1 (int): 第一个输入值 (0 或 1)。

            x2 (int): 第二个输入值 (0 或 1)。

        返回：

            int: XOR 运算的结果 (0 或 1)。

        """

        s1 = self.NAND(x1, x2)  *# 计算 NAND(x1, x2)*

        s2 = self.OR(x1, x2)  *# 计算 OR(x1, x2)*

        y = self.AND(s1, s2)  *# 计算 AND(NAND(x1, x2), OR(x1, x2))*

        return y  *# 返回 XOR 的结果*

*# 初始化感知机对象,该基于我的学号类实现了 AND, NAND, OR, XOR 逻辑运算*

c = Per202306630()

print("\*\*\*\*\*\*与非门NAND\*\*\*\*\*\*")  *# 打印 NAND 运算的标题*

print("0 0 -> " + str(c.NAND(0, 0)))  *# 测试 NAND(0, 0)*

print("1 0 -> " + str(c.NAND(1, 0)))  *# 测试 NAND(1, 0)*

print("0 1 -> " + str(c.NAND(0, 1)))  *# 测试 NAND(0, 1)*

print("1 1 -> " + str(c.NAND(1, 1)))  *# 测试 NAND(1, 1)*

print("\*\*\*\*\*\*AND\*\*\*\*\*\*")  *# 打印 AND 运算的标题*

print("0 0 -> " + str(c.AND(0, 0)))  *# 测试 AND(0, 0)*

print("1 0 -> " + str(c.AND(1, 0)))  *# 测试 AND(1, 0)*

print("0 1 -> " + str(c.AND(0, 1)))  *# 测试 AND(0, 1)*

print("1 1 -> " + str(c.AND(1, 1)))  *# 测试 AND(1, 1)*

print("\*\*\*\*\*\*OR\*\*\*\*\*\*")  *# 打印 OR 运算的标题*

print("0 0 -> " + str(c.OR(0, 0)))  *# 测试 OR(0, 0)*

print("1 0 -> " + str(c.OR(1, 0)))  *# 测试 OR(1, 0)*

print("0 1 -> " + str(c.OR(0, 1)))  *# 测试 OR(0, 1)*

print("1 1 -> " + str(c.OR(1, 1)))  *# 测试 OR(1, 1)*

print("\*\*\*\*\*\*XOR\*\*\*\*\*\*")  *# 打印 XOR 运算的标题*

print("0 0 -> " + str(c.XOR(0, 0)))  *# 测试 XOR(0, 0)*

print("1 0 -> " + str(c.XOR(1, 0)))  *# 测试 XOR(1, 0)*

print("0 1 -> " + str(c.XOR(0, 1)))  *# 测试 XOR(0, 1)*

print("1 1 -> " + str(c.XOR(1, 1)))  *# 测试 XOR(1, 1)*

运行结果

